CSE574 Introduction to Machine Learning

Machine Learning: Introduction

Jue Guo

University at Buffalo

January 25, 2024

Outline

- Logistics
- What is Machine Learning
- Types of Learning
  - Supervised Learning
  - Unsupervised Learning
  - Semi-Supervised Learning
  - Reinforcement Learning
- How Supervised Learning Works?
  - Supervised Learning Example: Spam Detection
  - Spam Detection: Learning Algorithm
  - Optimization Problem
  - SVM: 2D Example
  - Conclusion
- Why the Model Works on New Data?

About This Course

Instructor Information

- **Instructor**: Jue Guo
- **Office Hr**: Monday 9:00AM – 10:30AM
- **Location**: Zoom Link
- Read piazza post **frequently**.

Course Location and Time

- **Location**: Norton 190
- **Time**: Monday, Wednesday, and Friday, 3:00 pm to 3:50 pm.
- Drop Date: 1/31/24
- Attendance is not mandatory, and all the course note will be **handwritten** or through **slides** in class. Course material on GitHub can be used as a guide but **not** the ultimate contents being delivered in class.

Other Important Information

- Read FDOC Checklist on piazza. (25 minutes read)

What is Machine Learning?

Machine Learning is a sub-field of computer science concerned with building algorithms that, to be useful, rely on a collection of examples of some phenomenon.

- These examples can come from nature, be handcrafted by humans, or be generated by another algorithm.

Machine learning can also be defined as the process of solving a practical problem by

1. gathering a dataset
2. algorithmically building a statistical model based on that dataset.

The statistical model is assumed to be used somehow to solve the practical problem. To save keystrokes, “learning" and “machine learning" are used interchangeably.

Types of Learning

Learning can be supervised, semi-supervised, unsupervised, and reinforcement.

Supervised Learning

In **supervised learning**, the **dataset** is the collection of **labeled examples** $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. Each element $\mathbf{x}_i$ among N is called a **feature vector**.

- A feature vector is a vector in which each dimension $j = 1, ..., D$ contains a value that describes the example somehow. That value is called a **feature** and is denoted as $x^{(j)}$.
- The **label** y_i can be either an element belonging to a finite set of **classes** $\{1, 2, ..., C\}$, or a real number, or a more complex structure, like a vector, a matrix, a tree, or a graph. Unless otherwise stated, y_i is either one of a finite set of classes or a real number.

The goal of a **supervised learning algorithm** is to use the dataset to produce a **model** that takes a feature vector $\mathbf{x}$ as input and outputs information that allows deducing the label for this feature vector.

Unsupervised Learning

In **unsupervised learning**, the dataset is a collection of unlabeled examples $\{\mathbf{x_i}\}_{i=1}^{N}$. The goal of **unsupervised learning algorithm** is to create a **model** that takes a feature vector $\mathbf{x}$ as input and either transforms it into another vector or into a value that can be used to solve a practical problem.

- In **clustering**, the model returns the id of the cluster for each feature vector in the dataset.
- In **dimensionality reduction**, the output of the model is a feature vector that has fewer features than the input $\mathbf{x}$.
- In **outlier detection**, the output is a real number that indicates how $\mathbf{x}$ is different from a “typical” example in the dataset.

Semi-Supervised Learning

In **semi-supervised learning**, the dataset contains both labeled and unlabeled examples. Usually, the quantity of unlabeled examples is much higher than the number of labeled examples. The goal of a **semi-supervised learning algorithm** is the same as the goal of the supervised learning algorithm.

- using many unlabeled examples can help the learning algorithm to find a better model. It could look counter-intuitive that learning could benefit from adding more unlabeled examples. It seems we add more uncertainty to the problem. However, when you add unlabeled examples, you add more information about your problem: a larger sample reflects better the probability distribution the data we labeled came from.

Reinforcement Learning

Reinforcement learning is a subfield of machine learning where the machine “lives" in an environment and is capable of perceiving the *state* of the environment as a vector of features. The machine can execute *actions* in every state. Different actions bring different *rewards* and could also move the machine to another state of the environment. The goal of a **reinforcement learning algorithm** is to learn a *policy*.

- A policy is a function (similar to the model in supervised learning) that takes the feature vector of a state as input and outputs an optimal action to execute in that state. The action is optimal if it maximizes the *expected average reward*.

Reinforcement learning solves a particular kind of problem where decision making is sequential, and the goal is long term, such as game playing, robotics, resource management, or logistics. In our class, we will **not** include reinforcement learning.

How Supervised Learning Works?

Supervised learning is the type of machine learning most frequently used in practice. The supervised learning process starts with gathering the data. The data for supervised learning is a collection of pairs (input, output).

- *Input* could be anything, email messages, pictures, or sensor measurements.
- *outputs* are usually real numbers, labels (cat, dog, mouse), vectors (four coordinates), sequence or some other structure.

Spam Detection: Problem Definition

Problem Definition You gather the data, 10,000 email messages, each with a label either “spam" or “not_spam". Now you have to convert each email message into a feature vector. one common way to convert a text into a feature vector, called **bag of words**

Bag of Words

Dictionary: [“offer", “win", “free", “money", “hello", “meeting", “regards"]

- Email 1 (Spam): “Win free money now!”
    - "offer": 0, "win": 1, "free": 1, "money": 1, "hello": 0, "meeting": 0, "regards": 0
      This becomes [0, 1, 1, 1, 0, 0, 0]
- Email 2 (Non-Spam): “Hello, please confirm the meeting schedule. Regards”

However, the Dictionary will be a lot bigger and contains 20,000 alphabetically sorted words.

Learning Algorithm: SVM

Following the similar procedure, you now have machine readable input data, but the output labels are still in the form of human-readable text. Some learning algorithms require transforming labels into numbers.

- Some algorithms require numbers like 0 to represent "not_spam" and 1 as "spam"
- **Support Vector Machine** (SVM): positive label ("spam") has a numeric value of +1, and the negative label ("not_spam") has the value of -1.

At this point, you have a **dataset** and a **learning algorithm**, so you are ready to apply the learning algorithm to the dataset to get the **model**.

SVM sees every feature vector as a point in a high-dimensional space (20,000-dimensional). The algorithm puts all feature vectors on an imaginary 20,000 dimensional plot and draws an imaginary 19,999-dimensional line (a *hyperplane*) that separates examples with positive labels from examples with negative labels.

- In machine learning, the boundary separating the examples of different classes is called **decision boundary**.
- The equation of the hyperplane is given by two **parameters**, a real-valued vector **w** of the same dimensionality as our input vector **x**, and a real number *b*:

$$\mathbf{w}\mathbf{x} - b = 0 \tag{1}$$

  where the expression wx means $w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \ldots + w^{(D)}x^{(D)}$, and D is the number of dimensions of the feature vector **x**.

The predicted label for some input feature vector $\mathbf{x}$ is:

$$y = \mathsf{sign}(\mathbf{wx} - b) \tag{2}$$

where sign is a mathematical operator that takes any value as input and returns +1 if the input is a positive number or -1 if the input is a negative number. The goal of the **learning algorithm** – SVM in this case – is to leverage the dataset and find the optimal values $\mathbf{w}^*$ and b^*. Once the learning algorithm identifies these optimal values, the **model** $f(\mathbf{x})$ is then defined as:

$$f(\mathbf{x}) = \mathsf{sign}\left(\mathbf{w}^*\mathbf{x} - b^*\right) \tag{3}$$

Therefore, to predict whether an email message is spam or not spam using an SVM model, you have to take the text of the message, convert it into a feature vector, then multiply this vector by $\mathbf{w}^*$, subtract b^* and take the sign of the result. This will give us the prediction (+1 means "spam", -1 means "not_spam").

Optimization Problem

Now, how does the machine find $\mathbf{w}^*$ and b^*?

- It solves an **optimization problem**. Machines are good at optimizing functions under constraints.

So what are the constraints we want to satisfy here?

- First of all, we want the model to predict the labels of our 10,000 examples correctly. Remember that each example $i = 1, \ldots, 10000$ is given by a pair $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i$ is the feature vector of example i and y_i is its label that takes values either -1 or +1 .

So the constraints are naturally:

$$\begin{aligned} \mathbf{w}\mathbf{x}_i - b &\geq +1 \quad \text{if } y_i = +1 \\ \mathbf{w}\mathbf{x}_i - b &\leq -1 \quad \text{if } y_i = -1 \end{aligned}$$

We would also prefer that the hyperplane separates positive examples from negative ones with the largest **margin**.

- The margin is the distance between the closest examples of two classes, as defined by the decision boundary. A large margin contributes to a better generalization, that is how well the model will classify new examples in the future.
- To achieve that, we need to minimize the Euclidean norm of $\mathbf{w}$ denoted by $\|\mathbf{w}\|$ and given by $\sqrt{\sum_{j=1}^{D}\left(w^{(j)}\right)^2}$.

So, the optimization problem that we want the machine to solve looks like this:

- ***Minimize*** $\|\mathbf{w}\|$ ***subject to*** $y_i\left(\mathbf{w}\mathbf{x}_i - b\right) \geq 1$ ***for*** $i = 1, \ldots, N$.

The expression $y_i\left(\mathbf{w}\mathbf{x}_i - b\right) \geq 1$ is just a compact way to write the above two constraints. The solution of this optimization problem, given by $\mathbf{w}^*$ and b^*, is called the **statistical model**, or, simply, the model. The process of building the model is called **training**.

SVM: 2D Example

For two-dimensional feature vectors, the problem and the solution can be visualized as shown in the figure.

- The blue and orange circles represent, respectively, positive and negative examples, and the line given by $\mathbf{wx} - b = 0$ is the decision boundary.

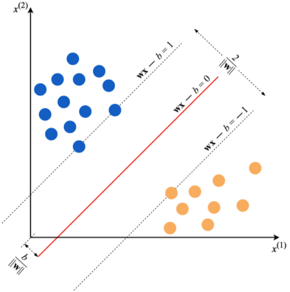

Figure: An example of an SVM model for two-dimensional feature vectors.

Why, by minimizing the norm of $\mathbf{w}$, do we find the highest margin between the two classes?

- Geometrically, the equations $\mathbf{wx} - b = 1$ and $\mathbf{wx} - b = -1$ define two parallel hyperplanes, as you see in figure. The distance between these hyperplanes is given by $\frac{2}{\|\mathbf{w}\|}$, so the smaller the norm $\|\mathbf{w}\|$, the larger the distance between these two hyperplanes.

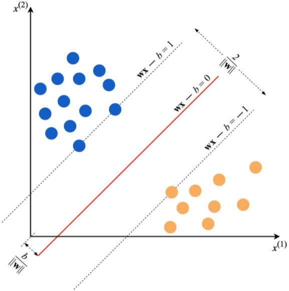

Figure: An example of an SVM model for two-dimensional feature vectors.

Conclusion

That's how Support Vector Machines work. This particular version of the algorithm builds the so-called linear model.

- It's called linear because the decision boundary is a straight line (or a plane, or a hyperplane). SVM can also incorporate **kernels** that can make the decision boundary arbitrarily non-linear.

In some cases, it could be impossible to perfectly separate the two groups of points because of noise in the data, errors of labeling, or **outliers** (examples very different from a "typical" example in the dataset). Another version of SVM can also incorporate a penalty hyperparameter[1] for misclassification of training examples of specific classes.

[1] A hyperparameter is a property of a learning algorithm, usually (but not always) having a numerical value. That value influences the way the algorithm works. Those values aren't learned by the algorithm itself from data. They have to be set by the data analyst before running the algorithm.

At this point, you should retain the following: ***any classification learning algorithm that builds a model implicitly or explicitly creates a decision boundary.***

- The decision boundary can be straight, or curved, or it can have a complex form, or it can be a superposition of some geometrical figures.
- The form of the decision boundary determines the **accuracy** of the model (that is the ratio of examples whose labels are predicted correctly).
- The form of the decision boundary, the way it is algorithmically or mathematically computed based on the training data, differentiates one learning algorithm from another.

In practice, there are two other essential differentiators of learning algorithms to consider: speed of model building and prediction processing time. In many practical cases, you would prefer a learning algorithm that builds a less accurate model quickly. Additionally, you might prefer a less accurate model that is much quicker at making predictions.

Why the Model Works on New Data?

Why is a machine-learned model capable of predicting correctly the labels of new, previously unseen examples?

- If two classes are separable from one another by a decision boundary, then, obviously, examples that belong to each class are located in two different subspaces which the decision boundary creates.

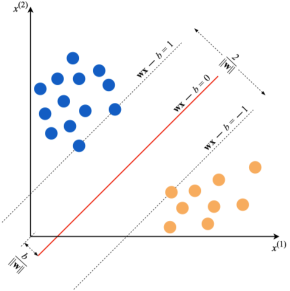

Figure: An example of an SVM model for two-dimensional feature vectors.

If the examples used for training were selected randomly, independently of one another, and following the same procedure, then, statistically, it is *more likely* that the new negative example will be located on the plot somewhere not too far from other negative examples.

- The same concerns the new positive example: it will *likely* come from the surroundings of other positive examples.

In such a case, our decision boundary will still, with *high probability*, separate well new positive and negative examples from one another.

- For other, *less likely* situations, our model will make errors, but because such situations are less likely, the number of errors will likely be smaller than the number of correct predictions.

Intuitively, the larger is the set of training examples, the more unlikely that the new examples will be dissimilar to (and lie on the plot far from) the examples used for training.

- To minimize the probability of making errors on new examples, the SVM algorithm, by looking for the largest margin, explicitly tries to draw the decision boundary in such a way that it lies as far as possible from examples of both classes.

Questions?